CodeWarrior®

Porting Guide

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 980829-JDR

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation 9801 Metric, Suite 100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Ordering	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	goto `Metrowerks`

Table of Contents

1 Introduction **7**

About The Porting Guide 7
Read the Release Notes! 7
What's New 7
Starting Points 8
Everyone Who Is New to CodeWarrior 8
Students and New Programmers 9
Command-line Programmers. 10
C and C++ Programmers 10
Turbo Pascal® Programmers 11
MPW™ Programmers 11
THINK™ C Programmers 11
THINK™ Pascal Programmers 11
A Note About Third Party Products 12

2 C/C++ Programming **13**

C/C++ Programming With CodeWarrior 13
Console I/O for Mac OS. 14
Command-Line Arguments for Mac OS 14
File Redirection for Mac OS 14
ARM and Other C++ Implementations 14
CFront and Metrowerks C++ 15
UNIX and POSIX Libraries 17
Including Files In C/C++ on Mac OS 18
The `sizeof()` operator and `int` 18
MSL Variations. 19

3 Pascal Programming **21**

Pascal Programming With CodeWarrior 21
Console I/O for Mac OS. 21
Turbo Pascal® I/O for Windows 22
Mac OS Toolbox Initialization 22
Pragmas and Directives 23
Working With Units 24

4 From Command-line to CodeWarrior 25
The Development Process 25
Make Files. 26
Editing 26
Compiling. 26
Linking 26
Debugging 27
Viewing Preprocessor Output 27
Command-line #define Directives 27
Scripting the IDE 27

5 Calling Mac OS Routines 29
Getting to the Mac OS 29
About Universal Interfaces 30
QuickDraw Global Variables. 30
Num2Str and Str2Num 31
HiWord and LoWord 31
68K and PowerPC Numerics. 32
SANE is Obsolete 32
Procedure Pointers, Callbacks, and UPPs 33
Referring to Universal Interfaces in Pascal 34
Referring to Universal Interfaces in C/C++ 35
Other Universal Interface Quirks 36

6 Cross-Platform Development 37
Cross-Platform Options 37

7 Troubleshooting 39
Troubleshooting Overview 39
Troubleshooting: Remember the release notes! 39
Troubleshooting: Solving “local data >32k” errors 39
Troubleshooting: C and Pascal Strings. 40
Troubleshooting: Solving Linker Errors 41
Troubleshooting: 12-byte Extended Floating Point Values . . 41
Troubleshooting: Assign() Doesn’t Work. 42
Troubleshooting: sizeof() returns a weird value. 42

Index **43**

1 Introduction

This guide offers hints and tips on moving your programming project from other platforms and development environments to CodeWarrior. It also points you to other CodeWarrior documentation for advanced topics.

About The Porting Guide

Use this guide if you are new CodeWarrior or you are converting a programming project from another development system to CodeWarrior.

This guide refers to development tools that might not be available with the CodeWarrior package you have. Consult the *Quick Start* guide that came with your CodeWarrior package for information on what's in your CodeWarrior package.

Read the Release Notes!

Before using CodeWarrior, read the information in the Release Notes folder, which is on the CodeWarrior CD-ROM and installed in the CodeWarrior folder on your computer's hard drive.

The release notes contain important information about new features, bug fixes, and any late-breaking changes.

What's New

This revision of the *Porting Guide* has been updated to include information for CodeWarrior for Windows users. Also, information that is specific to a particular computer platform is now more clearly identified.

New topics in the guide are:

- CFront and Metrowerks C++—discusses the differences beteween Metrowerks C++ and what is commonly called CFont.
- MSL Variations—minor variations between MSL C++ and *previous* versions of ANSI C++.

Starting Points

To help you get started, see these sections for information that you'll find useful:

- Troubleshooting Overview—To solve a problem quickly, see this section.
- Everyone Who Is New to CodeWarrior—Use this section if you've never used CodeWarrior before.
- Students and New Programmers—Use this section if you're just starting to learn how to program.
- Command-line Programmers—Use this section if you're used to using command-line shell environments like UNIX.
- C and C++ Programmers—Use this section if you want to develop a C or C++ programming project using CodeWarrior.
- Turbo Pascal® Programmers—Use this section if you're used to using Turbo Pascal tools.
- MPW™ Programmers—Use this section if you're used to using Apple's Macintosh Programmer's Workshop.
- THINK™ C Programmers—Use this section if you're used to using THINK C programming tools.
- THINK™ Pascal Programmers—Use this section if you're used to using THINK Pascal programming tools.

Everyone Who Is New to CodeWarrior

If you have never used CodeWarrior before, refer to the documentation listed here before using this guide. This guide assumes you already know how to use the CodeWarrior IDE.

- *Quick Start*—find out about getting started using CodeWarrior.
- *Tutorials*—learn how to use CodeWarrior to create, compile, link, run, and debug programs.
- *C, C++, And Assembly Language Reference*—the CodeWarrior C/C++ compilers and assemblers, their options, pragmas, and other details.
- *Pascal Language Manual*—how to use CodeWarrior Pascal. This manual also has reference material on the CodeWarrior implementation of the Pascal programming language.
- Students and New Programmers—what to do if you're using CodeWarrior as you learn about computer programming.
- Command-line Programmers—the differences between command-line software development and CodeWarrior's graphical user interface development environment.
- C and C++ Programmers—some important differences between Metrowerks C/C++ and other C/C++ compilers.
- Turbo Pascal® Programmers—what to do if you're used to using Turbo Pascal.
- MPW™ Programmers—the differences between the MPW environment and CodeWarrior.
- THINK™ C Programmers—the differences between THINK C™ and Metrowerks C and Metrowerks C++.
- THINK™ Pascal Programmers—the differences between THINK Pascal and Metrowerks Pascal.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

Students and New Programmers

If you are a student or a someone who is new to computer programming, refer to the documentation listed here.

These topics show you how to use CodeWarrior to create programs that use ordinary text input/output. These topics do not show you how to program a computer. Refer to "Everyone Who Is New to CodeWarrior" on page 8 for more information.

- C/C++ Programming With CodeWarrior—how to avoid headaches when using CodeWarrior C/C++.
- Pascal Programming With CodeWarrior—how to avoid frustration when using CodeWarrior Pascal.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

Command-line Programmers

If you are more familiar with UNIX™ or other command-line environments than CodeWarrior's graphical user interface environment, refer to these topics.

- The Development Process—explains important differences between the UNIX command line and the CodeWarrior IDE graphical user interface.
- C/C++ Programming With CodeWarrior—how to avoid frustration when using CodeWarrior C/C++.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

C and C++ Programmers

If you are porting a C or C++ program to CodeWarrior refer to these topics.

- C/C++ Programming With CodeWarrior—how to avoid frustration when using CodeWarrior C/C++.
- *MW C++ Lib Notes*—this manual has valuable information on some of the subtle differences between CodeWarrior's ANSI C++ library and other C++ libraries.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

Turbo Pascal® Programmers

If you are porting a Turbo Pascal program to CodeWarrior refer to these topics.

- Pascal Programming With CodeWarrior—how to avoid frustration when using CodeWarrior Pascal.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

MPW™ Programmers

If you are porting an MPW programming project to CodeWarrior, refer to these topics.

- The Development Process—explains important differences between the UNIX command line and the CodeWarrior IDE graphical user interface.
- *CodeWarrior Command-Line Tools* —this manual describes the MPW versions of the Metrowerks compilers, linkers, and other tools.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

THINK™ C Programmers

If you are porting a THINK C project to CodeWarrior refer to this information.

- C/C++ Programming With CodeWarrior—how to avoid frustration when using CodeWarrior C/C++.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

THINK™ Pascal Programmers

If you are porting a THINK Pascal project to CodeWarrior refer to this information.

- Pascal Programming With CodeWarrior—how to avoid frustration when using CodeWarrior Pascal.
- Troubleshooting Overview—this section lists topics on solving common problems when using CodeWarrior.

A Note About Third Party Products

The references to third party software development products in this guide are not endorsements for these products. Metrowerks makes no claims for the effectiveness, quality, or superiority of these products over products not mentioned in this guide.

There are many other third party products that might also be suitable, depending on your needs, but were not mentioned because of space constraints, appropriateness, availability, and so on.

2

C/C++ Programming

This chapter shows you how to port a C or C++ program to the CodeWarrior IDE (Integrated Development Environment).

C/C++ Programming With CodeWarrior

These topics discuss things to know about using CodeWarrior C/C++ compilers.

- Console I/O for Mac OS—creating programs that use simple text input/output.
- Command-Line Arguments for Mac OS—tips on converting programs that use command line arguments.
- File Redirection for Mac OS—tips on converting programs that use input/output redirection.
- ARM and Other C++ Implementations—discusses the differences between Stroustrup's *Annotated C++ Reference Manual* (ARM), and the CodeWarrior C++ implementation.
- CFront and Metrowerks C++—discusses the differences beteween Metrowerks C++ and what is commonly called CFont.
- UNIX and POSIX Libraries—discusses the differences between UNIX, POSIX, and the CodeWarrior implementations of the standard C and C++ libraries.
- The `sizeof()` operator and `int`—discusses a subtle difference between the type of value returned by `sizeof()` and the `int` data type.
- MSL Variations—discusses the differences between MSL C++ and *previous* version of ANSI C++

Console I/O for Mac OS

To convert a C or C++ program that uses simple text input/output and doesn't rely on any graphical user interface features of Mac OS refer to the *CodeWarrior Tutorials* manual.

For programs that use simple text input/output, CodeWarrior provides SIOUX. SIOUX is a Mac OS software package that automatically opens a text window, accepts characters from the keyboard, and handles menus. SIOUX does all this transparently; you, the programmer, do not have to explicitly invoke SIOUX.

Command-Line Arguments for Mac OS

The C/C++ `ccommand()` function provides a dialog box that allows the user to enter text as if it were at the command line. Refer to ccommand in the *C Library Reference* for more information.

File Redirection for Mac OS

For UNIX-style file redirection, use the C/C++ `ccommand()` function. Refer to ccommand in the *C Library Reference* for more information.

ARM and Other C++ Implementations

CodeWarrior C++ conforms to the ANSI C++ standard, although it has options to compile source code that conforms to the C++ specification in the *Annotated C++ Reference Manual* (ARM).

The ANSI Standards have changed the scope rules for a variable declared in a for loop. ARM allows the variable to stay in scope after the for loop block ends. The ANSI standards ends the scope of a variable when the for block ends (Listing 2.1).

Listing 2.1 Variable scope in for loop

```
for (int i: i<10; i++)
{
  // do stuff here
```

```
}

i = 3; // illegal, i is no longer in scope and must be re-declared
```

For information on compiling non-ANSI C++ source code, refer to the *C, C++, And Assembly Language Reference*.

CFront and Metrowerks C++

Metrowerks C++ is using the ANSI/ISO Proposed C++ standards for the C++ streams libraries, which will be referred to as ANSI C++. There are a few variations from the standards as accepted and published in such books as Stroustrup's *The C++ Programming Language, 2nd edition* (Addison-Wesly, 1991) and Stanley B. Lippman's *C++ Primer, 2nd edition* (Addison-Wesley, 1991) commonly referred to as the C++ Programming Language or CFront C++.

For information on compiling non-ANSI C++ source code, refer to the *C, C++, And Assembly Language Reference*.

#include header naming conventions

ANSI C++ proposed standards no longer require a file name extension for the include files. However, the file extension naming conventions are provided for by ANSI C++, and included in Metrowerks.

CFront C++

```
#include <iostream.h>
```

ANSI C++

```
#include <iostream>
```

File open modes

Listing 2.2 shows the valid combinations of `ios::openmode` for opening a file as defined in the ANSI C++ library—these are defined in terms of the equivalent `modestr` used in `fopen(s, modestr)` (see para 7.9.5.3 of ANSI Standard for C).

Listing 2.2 File open modes

```
ios::in   modestr = "r"
ios::out | ios::trunc  modestr = "w"
ios::out | ios::app  modestr = "a"
ios::in  | ios::bin  modestr = "rb"
ios::out | ios::trunc | ios::bin modestr = "wb"
ios::out | ios::app   | ios::bin modestr = "ab"
ios::in  | ios::out     modestr = "r+"
ios::in  | ios::out   | ios::trunc modestr = "w+"
ios::in  | ios::out   | ios::app modestr = "a+"
ios::in  | ios::out   | ios::bin modestr = "r+b"
ios::in  | ios::out   | ios::trunc | ios::bin modestr = "w+b"
ios::in  | ios::out   | ios::app   | ios::bin modestr = "a+b"
```

All other combinations are invalid and no file is opened and no error message is produced.

File open and close testing

Testing for an open file should be done with the `is_open()` function.

CFront C++

```
ofstream to(“testFile”0;
  if (!to )...
```

ANSI C++

```
ofstream to(“testFile”);
  if (to.is_open() == 0)...
```

iostream `fail()` vs. `eof()`

The description of iostreams approved by the committee is Revision 7. It makes a point of noting that the `eofbit` is set haphazardly in past practice. It also specifies most operations sufficiently tight that it’s hard to sneak in more consistent behavior and still conform to the draft standard. ANSI C++ libraries to not set the `eofbit`, therefore the previous practice of using the function `eof()`, which gave a non-zero result when the end of file was reached, is no longer useful. Instead you should test the value of the `fail()` function, which

will pick up both `eof` and other kinds of failure, as shown in Listing 2.3.

Listing 2.3 fail() vs. eof()

```
#include <iostream>
#include <fstream>

void main()
{
  // fill file with 10 x's
        ofstream out("testfile");
        for(int i = 0; i < 10; i++ ) out.put( 'X');
        out.close();

       char c = 0;
       ifstream input("testfile");

    //  while ( !input.fail() )   this leaves EOF character
    //  while( input.peek() != EOF )this works but less safe
  while ( !input.fail() && input.peek() != EOF )
       {
          c = input.get() ;
          cout << "char: " << c << endl;
      }
     input.close();
}
```

UNIX and POSIX Libraries

The CodeWarrior C/C++ libraries conform to the ANSI C and ANSI C++ standards.

There are, however, header files that provide some of the functions in the POSIX standard and many functions found in UNIX libraries that are not specified in the ANSI C or ANSI C++ standards.

Refer to `fnctl.h`, `stat.h`, `unistd.h`, `unix.h`, `utime.h`, and `utsname.h` in the *C Library Reference* for more information.

Including Files In C/C++ on Mac OS

When using CodeWarrior on Mac OS, CodeWarrior C/C++ handles subdirectory names in `#include` directives differently than UNIX compilers. In particular, Mac OS uses the colon, ":", as a directory separator character, not the slash, "/".

For example, when using CodeWarrior on a Mac OS computer, issuing this directive

```
#include "special/datatypes.h"
```

will actually include a file named "`special/datatypes.h`," not the file "`datatypes.h`" in the "special" directory.

Specify Subdirectories for #include Directives

The recommended way to specify a directory for an `#include` directive is to add the directory to the list of access paths in the project's **Access Paths** project settings panel and remove references to subdirectories in all `#include` directives. Refer to the *CodeWarrior IDE Guide* for more information on access path preferences.

Another way to specify a subdirectory for an `#include` directive is to convert the pathname to a Mac OS pathname. For example

```
#include "special/datatypes.h"
```

becomes

```
#include ":special:datatypes.h"
```

The `sizeof()` operator and `int`

When programming in C, don't assume that the value of a `sizeof()` operator is assignment compatible with the `int` or `long` data types.

According to the ANSI C standard `sizeof()` returns a value of type `size_t`, defined in `stddef.h`.

The `size_t` and `int` data types are often the same, but might differ depending on the platform or processor. Refer to Listing 2.4 for an

example of how a program executes incorrectly when it assumes that `size_t` and `int` are the same size.

Listing 2.4 Making assumptions about `sizeof()` and `int`

```
#include <stdio.h>
#include <stddef.h>

typedef struct {
  charbigArray[100000];
} MyStruct;

void main(void)
{
  int j;
  size_t t;


  j = sizeof(MyStruct); /* This doesn't work */
  t = sizeof(MyStruct); /* This works */

  printf("bad size of MyStruct = %ld\n", j);
  printf("good size of  MyStruct = %ld\n", t);
}

/* Output, running on Mac OS for 68K:

bad size of MyStruct = -2036334591
good size of  MyStruct = 100000

*/
```

MSL Variations

There are a few variations from the MSL implementation of C++ and *previous* versions of ANSI C++. MSL C++ conforms to the latest version of the ANSI C++ standard.

- The `fstream` class is now included for mixed input and output

- The file reading facilities `tellg()`, `tellp()`, `seekg()`, and `seekp()` are now included in the ANSI C++ standard
- The STL algorithms are now part of the proposed standard and conflict with older versions of the STL Libraries

3

Pascal Programming

This chapter shows you how to port a Pascal program to the CodeWarrior IDE (Integrated Development Environment).

Pascal Programming With CodeWarrior

These topics discuss things to know about using CodeWarrior Pascal compilers.

- Console I/O for Mac OS—tips on creating programs that use simple text input/output.
- Mac OS Toolbox Initialization—CodeWarrior Pascal doesn't initialize the runtime environment like other Pascal compilers.
- Turbo Pascal® I/O for Windows—How to make sure CodeWarrior Pascal supports Turbo Pascal routines and behaviors for input/output.
- Pragmas and Directives—Where to look for compiler directive information.
- Working With Units—CodeWarrior Pascal handles units and USES clauses differently than other Pascal compilers.

Console I/O for Mac OS

To create a Pascal program that uses simple text input/output and doesn't rely on any graphical user interface features of Mac OS or Microsoft Windows, refer to the *CodeWarrior Tutorials* .

For programs that use simple text input/output, CodeWarrior provides SIOUX. SIOUX is a software package that automatically opens a text window, accepts characters from the keyboard, and handles menus. SIOUX does all this transparently; you, the programmer, do not have to explicitly invoke SIOUX.

Turbo Pascal® I/O for Windows

When porting Pascal source code from Turbo Pascal to CodeWarrior Pascal that uses Turbo Pascal input/output routines, turn on the **Use Turbo Pascal I/O** checkbox in the Pascal Compiler panel in the Project Settings dialog box.

With this option on, CodeWarrior Pascal makes language extensions available that are compatible with the Turbo Pascal input/output model.

See the *Pascal Language Reference* for more information.

Mac OS Toolbox Initialization

Mac OS applications ported from other Pascal compilers to CodeWarrior Pascal for Mac OS sometimes crash immediately after they are launched. This "bug" can easily be fixed by properly initializing the Mac OS Toolbox.

Unlike other Pascal implementations, the runtime library for CodeWarrior Pascal for Mac OS does not automatically initialize the Toolbox. It's up to you, the programmer, to call the proper initialization routines before using QuickDraw, Window Manager, Menu Manager, and other managers in the Mac OS.

A typical initialization routine looks like Listing 3.1.

NOTE: You only need to initialize the Toolbox if your program calls the Mac OS Toolbox directly. Toolbox initialization isn't necessary if you don't use any Mac OS-specific features. See "Console I/O for Mac OS" on page 21 for more information.

Listing 3.1 Initializing the Mac OS Toolbox

```
(*
   Initialize the Toolbox before calling any
   other Mac OS routines.
*)
```

```
MaxApplZone;
MoreMasters;
InitGraf(@qd.thePort);
InitFonts;
InitWindows;
InitMenus;
TEInit;
InitDialogs(NIL);
FlushEvents(everyEvent, 0);
InitCursor;
```

See also

For more information on Toolbox initialization, refer to *Inside Macintosh: Overview*.

For information on programs that don't use the Toolbox, see "Console I/O for Mac OS" on page 21.

For other information on the Mac OS, see "Getting to the Mac OS" on page 29, "Referring to Universal Interfaces in Pascal" on page 34, and "QuickDraw Global Variables" on page 30.

Pragmas and Directives

CodeWarrior Pascal doesn't use pragmas like C/C++. Instead, source code generation is controlled by compiler directives.

Compiler directives are enclosed in a comment and preceded by a dollar sign ($). Only one directive can be in a comment.

An example of using a compiler directive is:

```
{$OV-} (* Turn off overflow checking *)
```

See the *Pascal Language Reference* for more information.

Working With Units

Unlike other Pascal implementations, CodeWarrior Pascal processes the `USES` clause based on the file name of the interface being included, rather than the name declared in its `UNIT` statement. The {$U} directive overrides this behavior and allows you to specify a different filename to use instead.

See also "Referring to Universal Interfaces in Pascal" on page 34.

4

From Command-line to CodeWarrior

This chapter shows you how to port a UNIX or MS-DOS programming project to the CodeWarrior IDE (Integrated Development Environment). Specifically, it describes the differences between command-line and CodeWarrior software development tools and methods.

The Development Process

For complete information on performing software development tasks like editing, compiling, and linking, refer to the *CodeWarrior IDE Guide*.

Read these topics to find out how using the CodeWarrior IDE differs from command-line programming.

- Make Files—The IDE's uses a project to control source file dependencies and settings for compilers and linkers.
- Compiling—How the IDE performs compile operations.
- Linking—How the IDE performs linking operations.
- Editing—A quick overview of source code editing from the IDE.
- Debugging—How to debug a program from the IDE.
- Command-line #define Directives—Where to find information on automatically inserting preprocessor instructions.
- Scripting the IDE—The IDE for Mac OS supports AppleScripting.
- Viewing Preprocessor Output—A tip on debugging preprocessor directives.

Make Files

The CodeWarrior IDE *project* is equivalent to the make file. The IDE lists all the files needed to create a program in the project. Among the kinds of files in a project are source code files and libraries. The IDE stores object code in the project, rather than in separate object code files. The IDE also stores the settings for compiler and linker options in the project.

For related information, see "Command-line #define Directives" on page 27, "Scripting the IDE" on page 27.

Editing

The CodeWarrior IDE has a text editor to edit source code. It handles text files in Mac OS, MS-DOS/Windows, and UNIX formats.

To edit a source code file, or any other editable file, that is in a project, just double-click the file's name in the project window.

Compiling

To compile a source code file, it must be in a project. Once it's in a project, just select it in the project window and choose **Compile** from the Project menu.

To compile all the files in the project that have been modified since they were last compiled, choose **Bring Up To Date** in the Project menu.

In UNIX and other command-line environments, object code compiled from a source code file is stored in a binary file (a ".o" or ".obj" file). The CodeWarrior IDE stores object in the project file itself; the IDE doesn't create separate object code files.

Linking

Linking object code into a final binary file is easy: use the **Make** command in the Project menu. The Make command brings the ac-

tive project up to date, then links the resulting object code into a final output file.

Unlike object code, which is stored inside the project, the final output file is stored in a file separate from the project. Use the **Project** preference panel to set the name of the final output file. Refer to the *CodeWarrior IDE Guide* for more information.

Debugging

To debug a project, make sure **Enable Debugging** is selected in the Project menu. To bring your project up to date, link it, and launch the final binary from the CodeWarrior Debugger, choose **Debug** in the Project menu.

For more information on debugging, refer to the *CodeWarrior Debugger Manual*.

Viewing Preprocessor Output

To view preprocessor output, select the file in the project window and choose **Preprocess** from the Project menu.

Command-line #define Directives

Although there is no direct way to specify command-line `#define` directives in a graphical user interface-based development system, the CodeWarrior IDE does have a work around: refer to the *CodeWarrior IDE Guide* for more information.

Scripting the IDE

The CodeWarrior IDE for Mac OS is scriptable through AppleScript, allowing you to automate tedious, complex, and time-consuming operations. Consult the *CodeWarrior IDE Guide* for more information.

5

Calling Mac OS Routines

This chapter gives a quick introduction to accessing the Mac OS through Apple's Universal Interfaces.

Getting to the Mac OS

Below are topics to help you access Mac OS routines and data structures.

- About Universal Interfaces—what they are, who produces them, why you should use them.
- QuickDraw Global Variables—how to access them through the Universal Interfaces.
- Num2Str and Str2Num—these routines aren't available for PowerPC. Read this topic to find out how to work around these routines.
- 68K and PowerPC Numerics—SANE isn't available for PowerPC. This topic describes the problem and offers a solution.
- Procedure Pointers, Callbacks, and UPPs—dealing with procedure pointers and the Universal Interfaces.
- HiWord and LoWord—a tip when using these Mac OS calls with CodeWarrior Pascal.
- Other Universal Interface Quirks—things to be aware of when using the Universal Interfaces.
- Referring to Universal Interfaces in Pascal—how to access the Universal Interfaces from CodeWarrior Pascal.
- Referring to Universal Interfaces in C/C++—how to access the Universal Interfaces from CodeWarrior C/C++.

About Universal Interfaces

CodeWarrior always comes with the latest Universal Interfaces, the API for the Mac OS. Apple produces and regularly updates the Universal Interfaces to fix bugs, accommodate new system software features, and anticipate future system software features.

The Universal Interfaces give identical access on both the 68K and PowerPC Mac OS platforms. With the Universal Interfaces, the same source code will give identical functionality when compiled for either variety of Mac OS computer.

Many of the differences between other compilers for Mac OS and CodeWarrior compilers are because of the changes the Universal Interfaces introduce. These changes are described in the following topics.

In C/C++ programming for Mac OS, the Universal Interfaces are often called the Universal Headers.

In Pascal programming for Mac OS, the Universal Interfaces also called the Universal Pascal Interfaces (UPIs).

For a list of related topics, see "Getting to the Mac OS" on page 29.

QuickDraw Global Variables

The Universal Interfaces require you to access the QuickDraw global variables through a global record variable called `qd`. Any references made to the Quickdraw globals must use the `qd` record. For example, the Pascal call

```
InitGraf(@thePort);
```

becomes

```
InitGraf(@qd.thePort);
```

In C, the call

```
InitGraf(&thePort);
```

becomes

```
InitGraf(&qd.thePort);
```

The QuickDraw data structures that must be accessed with the `qd` qualifier are: `randSeed`, `screenBits`, `arrow`, `dkGray`, `ltGray`, `gray`, `black`, `white`, `thePort`.

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

Num2Str and Str2Num

In Pascal, the SANE.p interface file provides `Num2Str` and `Str2Num` only when generating classic 68K object code. They are not supported in the Universal Interfaces when generating code for PowerPC or for CFM68K.

Instead, use the `Str2Dec`, `Dec2Num`, `Num2Dec`, and `Dec2Str` routines in the fp.p interface. These routines give more control over how character strings and floating point values are converted than do `Num2Str` or `Str2Num`.

If you can't replace `Num2Str` or `Str2Num` and must continue using them, you may write equivalent functions by using

- the fp.p routines `Str2Dec`, `Dec2Num`, `Num2Dec`, and `Dec2Str`.
- the Pascal runtime routines `StringOf` and `ReadString`.

NOTE: `Str2Dec` and `Dec2Str` in fp.p take C string parameters.

For related information see "About Universal Interfaces" on page 30, "68K and PowerPC Numerics" on page 32, and "SANE is Obsolete" on page 32.

HiWord and LoWord

CodeWarrior Pascal has two built-in functions, `HiWrd` and `LoWrd`, that provide identical functionality of the ToolUtils.p Universal Interface routines `HiWord` and `LoWord`, without the overhead of a ROM call. Use the `HiWrd` and `LoWrd` routines instead for better performance.

For information on these routines, refer to the *CodeWarrior Pascal Library Reference*.

The HiWrd and LoWrd routines are not available for CodeWarrior C/C++.

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

68K and PowerPC Numerics

For more information on porting 68K floating point operations and data types to PowerPC, refer to *Inside Macintosh:PowerPC Numerics*. Appendix A of this manual describes the differences between SANE and PowerPC numerics and offers tips on porting mathematical operations to PowerPC.

extended Data Type

The `extended` data type is not supported in PowerPC. Instead, use `double_t`, which is defined for both 68K and PowerPC. Refer to Appendix A of *Inside Macintosh: PowerPC Numerics* for more information.

See Also

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

For related information, see "Troubleshooting: 12-byte Extended Floating Point Values" on page 41.

For information on SANE, see "SANE is Obsolete" on page 32.

SANE is Obsolete

Consider SANE (Standard Apple Numerics Environment) obsolete when writing or updating software to be used on PowerPC-based Mac OS computers and on 68K-based Mac OS applications running under CFM 68K. SANE is the numerics environment Apple developed for "classic" 68K Macintosh computers.

SANE.h (for C/C++) and SANE.p (for Pascal) are no longer part of the Universal Interfaces. Instead, use the routines provided by fp.h (for C/C++) and fp.p (for Pascal).

See Also

See also "Num2Str and Str2Num" on page 31, "68K and PowerPC Numerics" on page 32.

Procedure Pointers, Callbacks, and UPPs

While ordinary procedure pointers work well when used only with 68K code, Universal Procedure Pointers (UPPs) are required when developing code for both 68K and PowerPC.

Specifically, UPPs allow the Mixed Mode Manager to handle 68K routine calls to PowerPC routines and PowerPC routine calls to 68K routines. UPPs are versatile, allowing you to combine PowerPC and 68K code without dealing with mode switches, parameter order on the stack, and other low-level issues. On the downside, converting source code to the PowerPC means explicitly creating a UPP every time you need to provide a callback routine to a Mac OS routine or any other piece of code that may either be in 68K or PowerPC object code.

Ways to create a new UPP are:

- call the `NewRoutineDescriptor` routine (in MixedMode.h for C/C++ and MixedMode.p for Pascal) with the appropriate parameters
- use one of the Universal Interfaces' declared routines to create a specific type of UPP.

To destroy a UPP:

- call the `DisposeRoutineDescriptor` routine (in MixedMode.h for C/C++, in MixedMode.p for Pascal)

For example, to create a new `ResErrUPP` in Pascal that you can pass to the Resource Manager you would call:

```
FUNCTION NewResErrProc(
  userRoutine: ResErrProcPtr): ResErrUPP;
```

passing a pointer to your procedure as the single parameter. `NewResErrProc` and other UPP-creating routines are implemented as glue code that is linked to your program automatically.

The Universal Interfaces provide routines to directly call universal procedure pointers. These routines are the interface to the Mixed Mode Manager.

A typical calling function might look like this:

```
PROCEDURE CallResErrProc(thErr: OSErr;
                         userRoutine: ResErrUPP);
```

When you are done with this UPP, you would dispose of it by calling

```
DisposeRoutineDescriptor(userRoutine);
```

These calls are only needed for writing a callback to a Mac OS routine that exists in a different architecture (PowerPC calling 680x0 or 680x0 calling PowerPC).

Note that these calling functions are only needed when you write code that expects a callback parameter from an outside source. Generally this is only required if your program allows loadable submodules or patches traps that expect callback functions. Thus you would only have to use the `CallResErrProc` procedure if you install a patch to the Resource Manager that needs to use the `ResError` callback routine.

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

Referring to Universal Interfaces in Pascal

CodeWarrior Pascal requires the explicit inclusion of a `USES` statement in every source code file that uses the Universal Interfaces. When moving from other Pascal compilers make sure to include all relevant units.

A `USES` statement for a typical program might look like this:

```
USES  Types, QuickDraw, Events, Windows,
      Dialogs, Fonts, DiskInit, TextEdit,
```

```
Traps, Devices, Memory, SegLoad,
Scrap, ToolUtils, OSUtils, Menus;
```

Instead of inserting this long clause in your source code, create a unit that uses all the Universal Interfaces your program requires. Include this unit in the USES clause in each of your project's source code files. Then select the **Uses Propagation** checkbox in the Pascal Language preferences panel.

NOTE: MPW Pascal users should note that the OSIntf and ToolIntf units are not available with the Universal Interfaces.

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

Referring to Universal Interfaces in C/C++

CodeWarrior C/C++ provides MacHeaders, a precompiled header file that includes the most-used headers in the Universal Interfaces.

The CodeWarrior stationery projects for Mac OS automatically use the appropriate MacHeaders file in the Prefix File field in the C/C++ Language preference panel. With this setting, `#include` directives for specific headers in the Universal Headers aren't even necessary.

Not all Universal Interface headers are precompiled in MacHeaders. MacHeaders projects contain the source code that includes the most-used Universal Interfaces in the MacHeaders precompiled header file. To modify the contents of MacHeaders, open the appropriate MacHeaders project and modify the source code files the project contains.

Refer to *Targeting Mac OS* for more information on MacHeaders.

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

Other Universal Interface Quirks

Many other routines, data structures, and variables have changed and continue to be renamed, added to, or removed from the Universal Interfaces. Apple makes these changes to fix bugs, to accommodate new system software features, and to help prepare developers for future system software releases.

To find out about changes to the Universal Interfaces for this CodeWarrior release, consult the Release Notes folder. Also, refer to "Troubleshooting: Solving Linker Errors" on page 41.

For more information on Universal Interfaces, see "About Universal Interfaces" on page 30.

6

Cross-Platform Development

This chapter gives tips on cross-platform development using CodeWarrior and third party tools.

Cross-Platform Options

There are a few ways to use CodeWarrior to use the same source code to generate applications for more than one computer platform. These topics discuss some of the options available.

ANSI C/C++ Development

Although not as flexible as other options, writing source code that strictly conforms to the ANSI C and ANSI C++ standards ensures that your source code will execute consistently on all the platforms for which CodeWarrior has a ANSI C or ANSI C++ library.

For more information on ANSI C and ANSI C++ conformance, refer to the *C, C++, And Assembly Language Reference*.

Third Party Products

There are many third party libraries and application frameworks that allow source code to be compiled for several different platforms.

Among the products that CodeWarrior tools support are:

- Altura Software—this company provides services to help developers port Mac OS applications to Microsoft Windows.
- c-tree Plus—a cross-platform database management library by FairCom.

See also "A Note About Third Party Products" on page 12.

7

Troubleshooting

This chapter gives tips on common problems encountered when developing with CodeWarrior.

Troubleshooting Overview

The topics covered by this chapter are:

- Troubleshooting: Remember the release notes!
- Troubleshooting: Solving “local data >32k” errors
- Troubleshooting: C and Pascal Strings
- Troubleshooting: Solving Linker Errors
- Troubleshooting: 12-byte Extended Floating Point Values
- Troubleshooting: Assign() Doesn’t Work
- Troubleshooting: sizeof() returns a weird value

Troubleshooting: Remember the release notes!

Read the release notes first if you can’t solve an obscure bug or error (“Read the Release Notes!” on page 7)

You’ll often find useful information, updates, and other news at the Metrowerks World Wide Web page:

```
http://www.metrowerks.com
```

Troubleshooting: Solving “local data >32k” errors

CodeWarrior C/C++ for Mac OS and CodeWarrior Pascal for Mac OS issue a “local data >32k” error when local data declared in a routine exceeds 32 Kbytes. The local data, often a declared array, is stored on the stack, which has a limit of 32Kbytes.

You can overcome this restriction by defining an array as static or using dynamic allocation to move the storage from the stack to the heap.

- Use standard memory allocation routines. In C/C++ use `malloc()` in stdlib.h or (non-standard) `alloca()` in alloca.h. In Pascal, use New().
- If your program runs on Mac OS, use Mac OS memory allocation routines like `NewHandle()` or `NewPtr()` in the Memory Universal Interface (Memory.h for C/C++, Memory.p for Pascal). For complete information on using Mac OS memory allocation routines see *Inside Macintosh: Memory*.

For more information on using Mac OS Universal Interfaces, see "Getting to the Mac OS" on page 29.

For more information on using library routines refer to the *C Library Reference*, the *C++ Library Reference*, and the *Pascal Library Reference*.

For more information on compiler and linker errors, refer to the *Errors Reference*.

For other troubleshooting topics, see "Troubleshooting Overview" on page 39.

Troubleshooting: C and Pascal Strings

Mac OS programming often requires converting character strings between the C and Pascal formats. C (and C++) character strings are terminated with a NUL (ASCII 0) character. Pascal character strings have an initial length field, followed by the actual character string.

The `c2pstr()`, `C2PStr()`, `p2cstr()`, and `P2CStr()` routines do C and Pascal string conversions.

These routines are in the Universal Interface TextUtils (TextUtils.h for C/C++ and TextUtils.p for Pascal).

See also "Working With Units" on page 24.

For other troubleshooting topics, see "Troubleshooting Overview" on page 39.

Troubleshooting: Solving Linker Errors

The linker issues "undefined '*symbol*'" errors when it cannot find symbol, an item that is referenced by the project. These errors typically happen when a routine or variable is misspelled or when a library is missing from the project.

The easiest way to avoid these errors is to create your new projects using the proper project stationery. Refer to the relevant language or target guide for information on which project stationery to choose when creating a new project.

If, even after using the proper project stationery, the linker continues to issue "undefined" errors then use the Find Library folder. The Find Library folder contains text files that list all the items in the libraries that come with CodeWarrior. Use the text files in this folder with the multi-file search feature in the CodeWarrior IDE **Find** dialog box to find the name of the library that's missing from your project.

For other troubleshooting topics, see "Troubleshooting Overview" on page 39 and "About Universal Interfaces" on page 30.

Troubleshooting: 12-byte Extended Floating Point Values

The Mac OS Universal Interfaces only support SANE's (Standard Apple Numerics Environment) 12-byte (96-bit) extended values for 68K code generation. To make your source code compatible for both 68K and PowerPC execution, use the fp Universal Interface (fp.h for C/C++, fp.p for Pascal).

For more information, see *Inside Macintosh: PowerPC Numerics*, Appendix A.

For related information, see "68K and PowerPC Numerics" on page 32 and "SANE is Obsolete" on page 32.

For other troubleshooting topics, see "Troubleshooting Overview" on page 39.

Troubleshooting: Assign() Doesn't Work

By default, CodeWarrior Pascal for Windows doesn't support the `Assign()` routine and other Turbo Pascal input/output routines. See "Turbo Pascal® I/O for Windows" on page 22 for more information.

Troubleshooting: sizeof() returns a weird value

In C or C++, the `sizeof()` operator sometimes seems to give a weird value if it is assigned to a variable or passed as an argument to a function. See "The sizeof() operator and int" on page 18 for more information.

Index

Symbols

30, 31, 32, 33
#define 27
#include 18
$ 23
$U 24
.o files 26
.obj files 26

Numerics

68K
    accessing Mac OS 30
    numerics 31, 32
    procedure pointers 33
    See alo PowerPC.
    size of int 19

A

alloca() 40
alloca.h 40
Annotated C++ Reference Manual 14
ANSI 14, 17, 18
ARM 14
arrow 31
ASCII 40
Assign() 42

B

black 31
Borland Pascal
    See Turbo Pascal.

C

C
    ANSI 17
    character strings 40
    starting points 10
C++
    ANSI 14
    ARM 14
    CodeWarrior 14
    starting points 10
C2PStr() 40
c2pstr() 40
callback routines 33
ccommand() 14
CFM68K 31
CFront 15
CodeWarrior
    contents 7
    for Windows 7
    getting started 8
    other development software 12
    package 7
CodeWarrior IDE
    getting started 8
command-line 10
console I/O 14, 21

D

Dec2Num 31
Dec2Str 31
#define 27
directive
    #define 27
    #include 18
    in Pascal 23
DisposeRoutineDescriptor 33
dkGray 31
double_t 32

E

-E 27
editing 26
extended 32, 41

F

files
    including 18
Find dialog box 41
Find Library folder 41
fnctl.h 17
fp.h 41
fp.p 41

G

gray 31

H

HiWord 31
HiWrd 31

I

I/O 14
#include 18
input 14, 21
input/output 14, 21
Inside Macintosh
    Memory 40
    Overview 23
    PowerPC Numerics 32, 41
int 18

L

Linker
    fixing errors 41
local data >32k 39
long 18
LoWord 31
LoWrd 31
ltGray 31

M

Mac OS
    initialization 22
    introduction 29
    Mixed Mode Manager 34
    Resource Manager 33
    Universal Interfaces 30
make file 26
malloc() 40
Memory.h 40
Memory.p 40
Menu Manager 22
Metrowerks
    WWW page 39
Microsoft Windows 11, 42
Mixed Mode Manager 34
MixedMode.h 33
MixedMode.p 33
MPW
    starting points 11
MS-DOS™ 10
MSL C++
    variations 19

N

new topics 8
New() 40
NewHandle() 40
NewPtr() 40
NewRoutineDescriptor 33
NUL characger 40
Num2Dec 31
Num2Str 31

O

OSIntf 35
output 14, 21

P

-P 27
P2CStr() 40
p2cstr() 40
Pascal
    character strings 40
    units 24
    USES clause 24
Porting Guide
    changes 8
    getting started 8
POSIX 17
PowerPC
    accessing Mac OS 30
    numerics 31, 32
    procedure pointers 33
    See also 68K.
pragmas 23
procedure pointers 33
ProcPtr 33

Q

Quick Start 7
QuickDraw 22, 30

R

randSeed 31
ReadString 31
redirection, file 14
release notes 39
Release Notes folder 7
Resource Manager 33

S

SANE 32, 41
screenBits 31
 30, 31, 32, 33
size_t 18
sizeof() 18, 42
source code 26
Standard Apple Numerics Environment 32
stat.h 17
stddef.h 18
stdio.h 19
stdlib.h 40
Str2Dec 31
Str2Num 31
StringOf 31
Stroustrup, Bjarne 14
students 9

T

text I/O 14, 21
TextUtils.h 40
TextUtils.p 40
thePort 31
THINK C
    starting points 11
THINK Pascal
    starting points 11
third party products 12
Toolbox 22, 29
ToolIntf 35
ToolUtils.p 31
topics, new 8
Turbo Pascal
    I/O 22, 42
    starting points 11

U

unistd.h 17
units 24
Universal Headers 30
Universal Interfaces 30
Universal Pascal Interfaces 30, 34
Universal Procedure Pointer 33
UNIX 10, 17
unix.h 17
UPI 30
UPP 33
Use Turbo Pascal I/O checkbox 22
USES clause 24
Uses Propagation checkbox 35
utime.h 17
utsname.h 17

W

white 31
Window Manager (Mac OS) 22
Windows
    See Microsoft Windows.
World Wide Web 39

CodeWarrior

Porting Guide

Credits

writing lead:	Carl B. Constantine
other writers:	Marc Paquette and Gavriel State
engineering:	Steve Chernicoff, Steve Jovanovic, Ron Liechty, Sarah Markey, Jeff Mattson, Jim Trudeau, and CodeWarrior users everywhere
frontline warriors:	CodeWarrior users everywhere

Guide to CodeWarrior Documentation

CodeWarrior documentation is modular, like the underlying tools. There are manuals for the core tools, languages, libraries, and targets. The exact documentation provided with any CodeWarrior product is tailored to the tools included with the product. Your product will not have every manual listed here. However, you will probably have additional manuals (not listed here) for utilities or other software specific to your product.

Core Documentation	
IDE User Guide	How to use the CodeWarrior IDE
Debugger User Guide	How to use the CodeWarrior debugger
CodeWarrior Core Tutorials	Step-by-step introduction to IDE components
Language/Compiler Documentation	
C Compilers Reference	Information on the C/C++ front-end compiler
Pascal Compilers Reference	Information on the Pascal front-end compiler
Error Reference	Comprehensive list of compiler/linker error messages, with many fixes
Pascal Language Reference	The Metrowerks implementation of ANS Pascal
Assembler Guide	Stand-alone assembler syntax
Command-Line Tools Reference	Command-line options for Mac OS and Be compilers
Plugin API Manual	The CodeWarrior plugin compiler/linker API
Library Documentation	
MSL C Reference	Function reference for the Metrowerks ANSI standard C library
MSL C++ Reference	Function reference for the Metrowerks ANSI standard C++ library
Pascal Library Reference	Function reference for the Metrowerks ANS Pascal library
MFC Reference	Reference for the Microsoft Foundation Classes for Win32
Win32 SDK Reference	Microsoft's Reference for the Win32 API
The PowerPlant Book	Introductory guide to the Metrowerks application framework for Mac OS
PowerPlant Advanced Topics	Advanced topics in PowerPlant programming for Mac OS
Targeting Manuals	
Targeting BeOS	How to use CodeWarrior to program for BeOS
Targeting Java VM	How to use CodeWarrior to program for the Java Virtual Machine
Targeting Mac OS	How to use CodeWarrior to program for Mac OS
Targeting MIPS	How to use CodeWarrior to program for MIPS embedded processors
Targeting NEC V810/830	How to use CodeWarrior to program for NEC V810/830 processors
Targeting Net Yaroze	How to use CodeWarrior to program for Net Yaroze game console
Targeting Nucleus	How to use CodeWarrior to program for the Nucleus RTOS
Targeting OS-9	How to use CodeWarrior to program for the OS-9 RTOS
Targeting Palm OS	How to use CodeWarrior to program for PalmPilot
Targeting PlayStation OS	How to use CodeWarrior to program for the PlayStation game console
Targeting PowerPC Embedded Systems	How to use CodeWarrior to program for PPC embedded processors
Targeting VxWorks	How to use CodeWarrior to program for the VxWorks RTOS
Targeting Win32	How to use CodeWarrior to program for Windows